

目錄

Spring Data

Spring Data 是spring 众多项目中的一个，但是我只看到了英文版，所以闲来无事自己有空就把它翻译成中文的英文原版在这里Spring Data JPA - Reference Documentation。 这个是这本书的github地址：https://github.com/tokyo2006/spring-data.git

前言

GitBook allows you to organize your book into chapters, each chapter is stored in a separate file like this one.

项目元数据

Version control - http://github.com/spring-projects/spring-data-jpa

Bugtracker - https://jira.spring.io/browse/DATAJPA

Release repository - https://repo.spring.io/libs-release

Milestone repository - https://repo.spring.io/libs-milestone

Snapshot repository - https://repo.spring.io/libs-snapshot

使用Spring Data Repositories

抽象Spring Data repository的目标是显著减少所需的数据访问层实现对各种持久存储的代码量。

> Spring Data repository 文档和你的模块。 这一章节解释了Spring Data repository 的核心概念和它的接口。 这一章的信息是从Spring Data通用模块取得。它使用的是Java Persistence API(JPA)模块的配置和示例代码。 你可以使用适用于XML的命名空间和声明来扩展特定模块。命名空间参考覆盖所有被支持repositoryAPI的Spring Data 模块和支持的XML配置, Repository 查询关键字覆盖查询支持通常的repository抽象方法关键字。对于你特殊功能的详细信息可以在相关模块的章节查看。

核心概念

在Spring Data repository 抽象的接口中心是仓库(Repository).它使得领域类的管理好比领域类的id类型就像一个类型参数，这个接口的行为主要是作为一个标记接库来捕获各种类型工作并帮助你发现接库并扩展它。CrudRepository为所管理的实体类提供复杂的CRUD功能。 事例1.CrudRepository 接口

```
public interface CrudRepository<T, ID extends Serializable> extends Repository<T,ID>{

    <S extends T> S save(S entity); //1

    T findOne(ID primaryKey);       //2

    Iterable<T> findAll();          //3

    Long count();                   //4

    void delete(T entity);          //5

    boolean exists(ID primaryKey);  //6

    // … more functionality omitted.
}
```

1.保存所给实例

2.返回所给ID的实例

3.返回所有实例

4.返回实例数量

5.删除所给实例

6.是否存在所给的ID实例

> 我们还提供了特别技术的持久性抽象,例如JpaRepository或者MongoRepository.这些接口扩展了CrudRepository和暴露了在基本持久技术中增加的潜在持久行技术接口，例如增删改查

在CrudRepository之上还有一个PagingAndSortingReporitory的抽象，它使得实体对象更容易进行分页

事例2. PagingAndSortingReposiory

```
public interface PagingAndSortingRepository<T, ID extends Serializable>
  extends CrudRepository<T, ID> {

  Iterable<T> findAll(Sort sort);

  Page<T> findAll(Pageable pageable);
}
```

访问每页20条数据第二页的用户信息你可以简单的这样做：

```
PagingAndSortingRepository<User, Long> repository = // … get access to a bean
Page<User> users = repository.findAll(new PageRequest(1, 20));
```

在增加的查询方法中，关键字的计数和删除查询都是有效的

事例3. 关键字计数查询

```
public interface UserRepository extends CrudRepository<User, Long> {

  Long countByLastname(String lastname);
}
```

事例4. 关键字删除查询

```
public interface UserRepository extends CrudRepository<User, Long> {

  Long deleteByLastname(String lastname);

  List<User> removeByLastname(String lastname);

}
```

查询方法

标准的CRUD功能的repositories通常在数据存储底层都有各种查询。使用Spring Data，声明这些查询有以下四步流程。

1. 声明一个继承于Repository的接口或者一个子接口并注入实体类和它的ID类型

```
interface PersonRepository extends Repository<User, Long> { … }
```

2. 在接口中声明查询方法

```
interface PersonRepository extends Repository<User, Long> {
List<Person> findByLastname(String lastname);
}
```

3. 使用Spring来为这些接口创建代理实例。也可以通过JavaConfig:

```
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@EnableJpaRepositories
class Config {}
```

或者通过XML配置:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:jpa="http://www.springframework.org/schema/data/jpa"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans.xsd
     http://www.springframework.org/schema/data/jpa
     http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">

   <jpa:repositories base-package="com.acme.repositories"/>

</beans>
```

JPA的命名空间也在这个例子中。如果你正在使用其他抽象repository，你需要改变相应的命名空间来替换jpa的声明支持你得存储模块，例如mongodb.并且注意JavaConfig的变量并不需要配置一个明确的包如同默认使用的声明类。定制包扫描。

4. 获取repository实例注入并使用它。

```
public class SomeClient {

  @Autowired
  private PersonRepository repository;

  public void doSomething() {
    List<Person> persons = repository.findByLastname("Matthews");
  }
}
```

以下的章节将会详细解释这些步骤

定义repository接口

首先定义一个特别的实体Repository interface。接口必须继承自Repository并且要定义为实体类和一个ID类型。如果你想要为这个实体类实现CRUD操作，继承CrudRepository替换Repository

调整repository定义

通常你的Repository interface将会继承自Repository,CrudRepository或者PagingAndSortingRepository。另外，如果你不想继承Spring Data的接口，你也能通过@RepositoryDefinition声明你的Repository interface。继承自CrudRepository会暴露一系列的方法来操作你的实体。如果你喜欢选择性的暴露某些方法，简单的从CrudRepository拷贝你想要暴露的方法到你的domain repository就可以了。

> 这允许你在Srping Data提供的功能上定义你自己的抽象。

示例7。选择性暴露CRUD方法

```
@NoRepositoryBean
interface MyBaseRepository<T ID extends Serializable> extends Repository<T,ID>{
    T findOne(ID id);
    T save(T entity);
}

interface UserRepository extends MyBaseRepository<User,Long>{
    User findByEmailAddress(EmailAddress emailAddress);
}
```

你首先为你所有的领域接口定义了一个通用的基础接口并且暴露了findOne(...)和save(...)方法。这些方法会路由到由你选择的Spring Data数据处理方式实现基础仓库中，例如JPA SimpleJapRepository,因为他们在CrudRepository匹配到了这些方法签名。所以UserRepository可以保存用户而且可以通过id找到用户，同样的也可以通过他们的邮件地址查询找到这些用户。

定义查询方法

Repository代理由两种方式从方法名字中导出特异性查询。它可以通过方法名直接导出查询，也能勇敢手动定义查询。可用选项依赖于实际的存储方式。然而这里通过策略的方式来决定哪些实际的查询被创建。让我们一起看看这些可用的选项。

查询查找策略

以下策略可用于仓库基础结构来解决查询。你可以在XML配置中的命名空间通过query-lookup-strategy属性来配置策略或者在JAVA配置中通过Enable${store}Repositories声明queryLookupStrategy属性。有些策略可能对于特别的datastores并不支持。

- CREATE 从查询方法名来尝试构建一个特别的数据查询。一般的方法都是从方法名称中移除已知设定好的前缀并且解析剩余的方法名。更多信息在创建查询
- USE_DECLARED_QUERY 尝试找到一个声明查询并在找不到的时候抛出一个异常。查询可以用声明的方式在任何地方进行定义或者被其他方法声明。查看特别的store文档来找到对这个store可用的选项.如果载入时间中在基础repository中找不到为这个方法这个声明的查询，这个查询就失败了。
- CREATE_IF_NOT_FOUND (默认)结合CREATE和USE_DECLARED_QUERY。它会首先查找声明过的查询，如果找不到被声明的查询，它会创建一个以方法名为基础的自定义查询。这就是默认的查找策略，因此如果你没有明确的做任何配置就会使用默认的策略。它能够让你通过方法名快速的查询定义但也可以通过引入所需的声明查询自定义调整这些查询。

创建查询

基础Spring Data repository内置的查询生成器机制对于创建实体仓库的约束查询是有用的，它会从方法名中去掉find...By,read...By,query...By,count...By和get...By这些前缀并解析剩下的内容.这些前缀还能包含更多的表达式例如Distinct,设置一个distinct标志并在查询中创建它，然后第一个By的动作就像一个分隔符来表明查询实际标准的开始。最基本的方式你可以在实体属性中定义表达式并用AND和OR连接它们。

```
public interface PersonRepository extends Repository<user,Long>{
    List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

    //Enables the distinct flag for the query
    List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname,String firstname);

    List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

    // Enabling ignoring case for an individual property
    List<Person> findByLastnameIgnoreCase(String lastname);
    // Enabling ignoring case for all suitable properties
    List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

    // Enabling static ORDER BY for a query
    List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
    List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
}
```

解析方法实际的结果依赖于你创建查询的持久存储。然后这里还算是有一些需要注意的事情。

- 表达式通常会结合级联的操作来进行属性遍历。你可以结合表达式属性AND和OR。对于属性表达式你也可以使用可支持的操作比如Between,LessThan,GreaterThan ,Like。这些支持的操作由不同的数据存储而不同，所以你需要查看你参考的文档中合适的部分。
- 方法解析支持为某些属性设置一个IgnoreCase标志(例如，findByLastnameIgnoreCase(...))或者为所有属性都支持忽略类型(通常是String实例，例如，findByLastnameAndFirstnameAllIgnoreCase(...))。是否支持ignoring cases 可能根据store不同而不同，所以相关部分在特殊库查询方法的参考文档中。
- 你可以通过增加一个OrderBy字段在按照属性来升降序的查询方法中用来静态排序。想要创建一个查询方法能够支持动态排序，请看特殊参数处理

属性表达式

正如前面的例子所示,属性表达式只能引用托管实体的直接属性。在查询创建的时候你已确认解析的属性是托管实体类的属性之一。然而你也能通过遍历嵌套属性来定义约束。假设一个Person拥有含ZipCode的Address。在这个方法名称中

```
List<Person> findByAddressZipCode(ZipCode zipCode);
```

创建了一个属性遍历x.address.zipCode。解析算法由理解实体的部分(AddressZipCode)为属性开始并检查与属性同名(未大写)的领域类。如果解析成功则使用那个属性，如果不成功，算法则会按照驼峰法则从右边开始将源拆开为头和尾并找到相应的属性，在我们的实例中，AddressZip和Code。如果算法从头那找到一个属性它会从尾部开始并继续解析，按照上面的方式来拆解尾部。如果第一个拆解不匹配，那么算法会移动到左边的拆分点(Address,ZipCode)继续分析。

虽然这个工作可能会有很多可能，它也可能因为算法选择错误的属性。假设Person类有一个addressZip属性，则算法会匹配第一个分拆点并且基本上会选择一个错误的属性并失败(就像addressZip类型属性没有code属性)。

要解决这种歧义你可以使用_加在你的方法名字中来手动的定义遍历节点。所以我们的方法名称会看起来像这样：

```
List<Person> findByAddress_ZipCode(ZipCode zipCode);
```

我们将下划线作为保留字符 我们强烈建议遵循标志的JAVA命名规范(比如禁止使用下划线在属性命名中而是用驼峰法则替代)。

特殊参数处理

正如上面所看到的那样，在您的查询中处理参数，您只需定义方法参数。除了基本的还要识别某些特别的类型像Pageable和Sort这些在你的查询中提供动态的分页和排序。

示例9。使用Pageable,Slice和Sort在查询方法中

```
Page<User> findByLastname(String lastname, Pageable pageable);

Slice<User> findByLastname(String lastname, Pageable pageable);

List<User> findByLastname(String lastname, Sort sort);

List<User> findByLastname(String lastname, Pageable pageable);
```

第一个方法允许在你的查询方法的静态定义查询中通过一个org.springframework.data.domain.Pageable实例来动态的添加分页。分页类知道元素的总数和可用页数。它通过基础库来触发一个统计查询计算所有的总数。由于这个查询可能对store消耗巨大，可以使用Slice来替代。Slice仅仅知道是否有下一个Slice可用，这对查询大数据已经足够了。

排序选项和分页的处理方式一样。如果你需要排序，简单的添加一个org.springframework.data.domain.Sort参数到你的方法即可。也正如你所见，简单的返回一个列表也是可以的，在这种情况下，生产的分页实例所需的附加元数据将不会被创建(这也意味着额外的计数查询可能需要但不一定被公布)。

> 要找到在你的查询中有多少页，你需要触发一个额外的计数查询。按照默认来说这个查询可以从你实际触发查询中衍生出来

限制查询

查询方法的结果可以通过关键字first或者top来限制,它们可以交替使用。在top/firest后添加数字来表示返回最大的结果数。如果没有数字，则默认假定1作为结果大小。

示例10。用Top和First查询限制结果大小

```
User findFirstByOrderByLastnameAsc();

User findTopByOrderByAgeDesc();

Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);

Slice<User> findTop3ByLastname(String lastname, Pageable pageable);

List<User> findFirst10ByLastname(String lastname, Sort sort);

List<User> findTop10ByLastname(String lastname, Pageable pageable);
```

限制表达式也支持Distinct关键字。对于限制查询的结果集定义到一个实例中包装这个结果到一个Optional中也是被支持的。

如果分页或者切片被应用到一个限制查询分页(计算多少页可用)则它也能应用于限制结果。

> 要注意结合通过Sort参数动态排序的限制结果容许表达查询的方法为“K”最小的，以及“K”最大的元素。

流查询结果

查询方法能对以JAVA 8的Stream为返回的结果进行逐步处理。而不是简单地包装查询结果在被用来执行流的流数据存储特定的方法。

示例11。以JAVA 8的Stream来进行查询的流处理结果

```
@Query("select u from User u")
Stream<User> findAllByCustomQueryAndStream();

Stream<User> readAllByFirstnameNotNull();

@Query("select u from User u")
Stream<User> streamAllPaged(Pageable pageable);
```

> 一个数据流可能包裹底层数据存储特定资源，因此在使用后必须关闭。 你也可以使用close()方法或者JAVA 7 try-with-resources区块手动关闭数据流。

示例12.在try-with-resources块中操作一个Stream

```
try(Stream<User stream = repository.findAllByCustomQueryAndStream()){
    stream.forEach(...);
}
```

> 当前不是所有的Spring Data模块都支持Stream作为返回类型

创建repository实例

在这个部分你创建实例和为repository接口定义的bean。这样做的一个方法是使用Spring的名称空间，这是与每个Spring Data模块，支持存储机制，虽然我们一般建议使用的JAVA配置风格的配置。

XML配置

每一个Spring Data模块都包含repositories元素能够让你简单的基于base-package定义来进行Spring扫描。

示例13。 通过XML来开启Spring Data repositories

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns:beans="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.springframework.org/schema/data/jpa"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">

  <repositories base-package="com.acme.repositories" />

</beans:beans>
```

JavaConfig

你也可以在一个JavaConfig类中使用 `@Enable${store}Repositories` 声明来触发repository的构建。基于java类配置的spring容器介绍请参考此文档JavaConfig in the Spring reference documentation 一个简单的开启Spring Data repositories的配置看上去是这样的

```
@Configuration
@EnableJpaRepositories("com.acme.repositories")
class ApplicationConfiguration {

  @Bean
  public EntityManagerFactory entityManagerFactory() {
    // …
  }
}
```

> 示例使用了JPA声明，你也可以根据你实际使用的store模块替换掉这个声明，这同样也适用于定义的EntityManagerFactory bean.The same applies to the definition of the EntityManagerFactory bean. 你可以查阅不同的store实现配置。

独立使用

你也可以在spring容器外使用repository组件，比如在CDI环境中，你依然需要一些spring的libraries在你的classpath中，但通常你也能以编程的方式来搭建repositories。你可以像下面示例一样使用由Spring Data模块提供各种repository持久化支持的RepositoryFactory。

```
RepositoryFactorySupport factory = … // Instantiate factory here
UserRepository repository = factory.getRepository(UserRepository.class);
```

自定义实现Spring Data repositories

有时候需要自定义实现一些repository的新方法。Spring Data repositories允许你很容易集成你自己实现的repostiroy功能，包括抽象的CRUD和查询功能

为单一的repositories添加自定义行为

为了给repository添加更丰富的自定义功能，首先你需要定义一个接口和实现这个接口中的方法。使用你提供的repostiroy接口来扩展自定义接口

Example 22. Interface for custom repository functionality

```
interface UserRepositoryCustom {
  public void someCustomMethod(User user);
}
```

Example 23. Implementation of custom repository functionality

```
class UserRepositoryImpl implements UserRepositoryCustom {

  public void someCustomMethod(User user) {
    // Your custom implementation
  }
}
```

> 在实现的类中一定要加上**Impl**这个后缀，这点很重要

实现类本身并没有依赖任何的Spring Data，所以实现类也是一个正常的spring bean,所以你可以使用标准的依赖注入行为将它注入到其它bean中比如jdbcTemplate,诸如此类。

Example 24. Changes to the your basic repository interface

```
interface UserRepository extends CrudRepository<User, Long>, UserRepositoryCustom {

  // Declare query methods here
}
```

让你的标准repository扩展为一个自定义的。组合CRUD和自定义的方法并让其在客户端可用。

配置

如果你使用命名空间配置，repository构件会在定义的类包中自动扫描自定义实现。这些自定义类必须按照repository-impl-postfix的命名规则命名，默认的后缀名为Impl

Example 25. Configuration example

```
<repositories base-package="com.acme.repository" />

<repositories base-package="com.acme.repository" repository-impl-postfix="FooBar" />
```

第一个配置示例会查找类com.acme.repository.UserRepositoryImpl来作为自定义repository的实现类，而第二个示例则会尝试查找com.acme.repository.UserRepositoryFooBar

手动连接

上面的示例展示了定义实现使用基础声明的自动连接，如何其它spring bean一样。如果自定义实现需要一些特别的连接，仅需要按照规则简单的声明和命名，构建就会参考手动定义的bean名字，而不是自动创建一个。

Example 26. Manual wiring of custom implementations

```
<repositories base-package="com.acme.repository" />

<beans:bean id="userRepositoryImpl" class="…">
  <!-- further configuration -->
</beans:bean>
```

为所有的repositories添加自定义行为

之前的章节并没有实现当你想把一个方法添加到所有的repository接口中。要添加一个自定义行为到所有的repository中，你首先需要添加一个中介接口来声明一个共享的行为。

Example 27. An interface declaring custom shared behavior

```
@NoRepositoryBean
public interface MyRepository<T, ID extends Serializable>
  extends PagingAndSortingRepository<T, ID> {

  void sharedCustomMethod(ID id);
}
```

现在你自己的repository需要扩展这个中介接口来替换之前包含方法声明的Repository接口，接着创建一个扩展持久化repository基础类的中介接口的实现类，这个类之后会作为代理自定义的repository基础类。

Example 28. Custom repository base class

```
public class MyRepositoryImpl<T, ID extends Serializable>
  extends SimpleJpaRepository<T, ID> implements MyRepository<T, ID> {

  private final EntityManager entityManager;

  public MyRepositoryImpl(JpaEntityInformation entityInformation,
                          EntityManager entityManager) {
    super(entityInformation, entityManager);

    // Keep the EntityManager around to used from the newly introduced methods.
    this.entityManager = entityManager;
  }

  public void sharedCustomMethod(ID id) {
    // implementation goes here
  }
}
```

Spring Data的扩展

网络支持

Repository填充器

保留的网络支持

JPA Repositories

介绍

Spring命名空间

基于配置的声明

持久化实体

保存实体

查询方法

查询查找策略

创建查询

使用JAP命名查询

使用@Query

使用命名参数

使用SpEL表达式

修改查询

应用查询提示

配置Fetch和LoadGraphs

使用存储过程

说明

事务处理

事务处理查询方法

锁定

审查

基本审查

JPA审查

审查配置

持久化单元的合并

路径扫描@Entity类和JPA映射文件

创建Repository实例集成

附录A:命名空间参考

附录B:填充命名空间参考

附录C:Repository查询关键字

附录D:常见问答